Vol.2019-CSEC-85 No.17
Vol.2019-IOT-45 No.17
2019/5/24

Secure and Compact Elliptic Curve Cryptosystems

YAOAN JIN[1] ATSUKO MIYAJI[1]

Abstract: Elliptic curve cryptosystems (ECCs) are widely used because of their short key size. They can ensure enough security with shorter keys, and use less memory space to reduce parameters. Hence, an elliptic curve is typically used in embedded systems. The dominant computation of an ECC is scalar multiplication $Q = kP, P \in E(F_q)$. Thus, the security and efficiency of scalar multiplication are paramount. To render secure ECCs, complete addition formulae can be employed for a secure scalar multiplication. However, this requires significant memory and is thus not suitable for compact devices. Several coordinates exist for elliptic curves such as affine, Jacobian, projective. The complete addition formulae are not based on affine coordinates and thus require considerable memory. In this study, we achieved a compact ECC by focusing on affine coordinates. In fact, affine coordinates are highly advantageous in terms of memory but require many `if statements` for scalar multiplication owing to exceptional points. We improve the scalar multiplication and reduce the limitations for input k. Furthermore, we extend the affine addition formulae to delete some exceptional inputs for scalar multiplication. Our compact ECC reduces memory complexity up to 26 % and is much more efficient compared to Joye's RL 2-ary algorithm with the complete addition of formulae when the ratio I/M of computational complexity of inversion (I) to multiplication (M) is less than 7.2.

Keywords: Elliptic curve scalar multiplication, side channel attack(SCA), exception-free addition formulae

1. Introduction

Elliptic curve cryptosystems (ECCs) are widely used because of their short key size. They can ensure enough security with shorter keys, and use less memory space to reduce parameters. Hence, an elliptic curve is typically used in embedded systems [1]. The dominant computation of ECCs is scalar multiplication $Q = kP, P \in E(F_q)$. Thus, the security and efficiency of scalar multiplication is paramount.

Studies regarding secure elliptic curve scalar multiplication algorithms can be divided into two. One pertains to prior studies regarding efficient secure scalar multiplication [6], [7], [8], [10]. The other pertains to efficient coordinates with addition formulae. Several coordinates for elliptic curves exist such as affine, Jacobian, and projective. Although it appears that we need to only combine efficient secure scalar multiplication with efficient coordinates, it is in fact not that simple because some scalar multiplications require branches to apply the addition formulae. For example, in the case of affine or Jacobian coordinates, both doubling and addition formulae exist for two inputs of P and Q [4]. That is, when the scalar multiplication algorithm employs addition formulae in affine or Jacobian coordinates, we need to versify whether the two input points are equal. In fact, not only the condition $P = Q$ but also other points such as $O + P$, $P - P$, and $2P = O$ become exceptional inputs. Hence, researchers have investigated on complete addition formulae [5], [11], [13], which can compute for any two input points. Further, new methods have been proposed by combining a powering ladder with complete addition formulae to protect the elliptic curve scalar multiplication from side channel attack (SCA) [12].

Complete addition formulae operate well to exclude such branches. However, complete addition formulae are not efficient from the memory and computational standpoints. Particularly, complete addition formulae are not based on affine coordinates and thus require significant memory.

In this study, we achieved a compact ECC by focusing on affine coordinates. In fact, affine coordinates are highly advantageous in terms of memory but requires many `if statements` for scalar multiplication owing to exceptional points. We adopt two approaches. First, we analyze a scalar multiplication with the input point and scalar k in detail by assigning three notions of generality of k, secure generality, and executable coordinate. Subsequently, we demonstrate that the Montgomery ladder[8], Joye's LR 2-ary algorithm [7], and Joye's RL 2-ary algorithm [7] satisfy the secure generality but that Joye's double-add algorithm[6] does not satisfy secure generality. Further, we verify coordinates that becomes executable. Subsequently, we improve Joye's RL 2-ary algorithm [7] to reduce the limitations for input k. Further, we extend the affine addition formulae to delete some exceptional inputs for scalar multiplication. Subsequently, we propose a new scalar multiplication by combining our improved Joye's RL 2-ary algorithm to our extended affine addition formulae. We enhance the efficiency of our method by 2-bit scanning using the affine double and quadruple formulae (DQ) [9], that can compute both $2P$ and $4P$ simultaneously with only one inversion computation. Finally, our compact ECC reduces memory complexity by 36 % and is more efficient compared to Joye's RL 2-ary algorithm with complete addition formulae when the ratio of inversion to multiplication is less than 7.2.

This paper is organized as follows. We first introduce the re-

[1] Graduate School of Engineering Osaka University

lated work in Section 2. In Section 3, we analyze a scalar multiplication from the point of input scalar k in detail assigning three new notions. Subsequently, we propose a variant of the affine addition formulae in Section 4. We improve Joye's RL 2-ary algorithm to reduce the limitations for input k and coordinates in Section 5. We compare our scalar multiplication with the affine formulae to previous scalar multiplication algorithms with complete addition formulae in Section 6. We conclude our work in Section 7.

2. Related work

The related studies regarding secure elliptic curve scalar multiplication algorithms can be divided into two. One pertains to prior studies regarding efficient scalar multiplication [6], [7], [8], [10] and the other pertains to efficient complete addition formulae [5], [11], [13]. Some scalar multiplications require branches to apply the addition formulae. Complete addition formulae operate well to exclude such branches. However, complete addition formulae are not efficient from the memory and computational standpoints. We focus on right-to-left (RL) algorithm in this paper.

2.1 Scalar multiplication

Montgomery ladder scanning scalar from MSB to LSB without dummy computations can compute scalar multiplications regularly [8]. Thus, in the Montgomery ladder, the security issue depends on the addition formulae of the elliptic curve. If we utilize addition formulae on affine or Jacobian coordinates, branches to avoid additions on two inputs exist, such as $P + P$, $P - P$, and $O + P$, and the doubling of P with $2P = O$. Branches results in SCA. Hence, upon implementation, we should use "if statements" carefully. Meanwhile, if we utilize complete addition formulae [11], then we exclude "if statements" but sacrifice memory and computational efficiency [12].

As for Joye's double-add algorithm, Algorithm 1, by scanning a scalar from LSB to MSB [6], the same discussion as the above holds. Furthermore, for regular right-to-left (RL) m-ary, Algorithm 2 are proposed in [7]. In this m-ary algorithm, the same discussion as that of the Montgomery ladder holds. It is noteworthy that both the regular LR m-Ary and RL m-Ary algorithms are suitable for scalar multiplications with m-Ary representation. The regular 2-Ary algorithms are improved from Algorithm 2 by assuming that the MSB of the input scalar is always '1' in [7]. However, they can not compute scalar multiplications correctly when the scalar begins with '0'. All of these ladders are regular and without dummy computations. They perform equally well compared to Montgomery ladder mentioned before.

2.2 Complete Elliptic Curve Addition formulae

Izu and Takagi proposed the x-only differential addition and doubling formulae [5], which proved to be exceptional only if both input coordinates of x and z are 0 [12]. These addition formulae are applied to the Montgomery ladder, in which after the computation of the x-coordinate, the y-coordinate can be recovered by the formula of Ebeid and Lambert [3].

Renes, Joost, Craig Costello, and Lejla Batina proposed complete addition formulae for prime order elliptic curves [11]. Based on the theorems of Bosma and Lenstra [2], the complete addition formulae for an elliptic curve $E(\mathbb{F}_p)$ can be obtained without points of order two. $E(\mathbb{F}_p)$ with prime order excludes the points of order two, thus, we can use the complete addition formulae on $E(\mathbb{F}_p)$. The authors also mentioned that if the complete addition formulae were used in an application, their efficiency could be improved based on specific parameters and further computation. However, they are still costly.

Table 1 summarizes the addition formulae including the complete addition formulae, where M, S, I, and A are the costs for one field multiplication, square, inversion and addition, respectively; further, ma and mb are the costs for multiplication to a and b, respectively,

Assuming that $S = 0.8M$ and ignoring the computational complexity of ma, mb, and A, the computational complexity of ADD + DBL in complete addition is $24M$. Subsequently, the computational complexity of ADD + DBL in affine is more efficient than that in complete addition or Jacobian when $I < 8.8M$ or $I < 8.2M$. Meanwhile, the computational complexity of ADD + DBL in Jacobian is always more efficient than that in complete addition by $11.2M$.

Table 1 Computational complexity of Elliptic Curve Addition Formulae

Method	Conditions	ADD	DBL	Memory
x-only addition[5]	Either x or z-coordinate is not 0	$8M + 2S$	$5M + 3S$	10
Complete addition[11]	$2 \nmid \#E(\mathbb{F}_p)$	$12M+3ma+2mb+23A$	$12M+3ma+2mb+23A$	15
Affine	-	$2M + S + I$	$2M+2S+I$	5
Jacobian	-	$12M + 4S$	$2M + 7S$	8

3. Exceptional inputs in scalar multiplication

This section analyzes two algorithms (Algorithms 1–2) with input scalar $k = \sum_{i=0}^{\mathfrak{l}-1} k_i 2^i$ (in binary) and point P from the following three aspects: generality of k, secure generality, and executable coordinate.

3.1 Generality of k

We define the *generality* of k as follows. The scalar multiplication should compute kP for $\forall k \in [0, N-1]$, where $N \in \{0,1\}^{\mathfrak{l}}$ is the order of P. Subsequently, it includes a case where the MSB of k is zero ($k_{\mathfrak{l}-1} = 0$). We say that a scalar multiplication satisfies the generality if it can operate for any $k \in [0, N-1]$ with ($k_{\mathfrak{l}-1} = 0$) or ($k_{\mathfrak{l}-1} = 1$). Let us investigate whether Algorithms 1–2 satisfy the generality of input scalar k. The Joye's double-add algorithm (Algorithm 1) can operate for any input scalar $k \in [0, N-1]$. It is obvious that Algorithm 1 can compute kP correctly when $k_{\mathfrak{l}-1} = 1$. Algorithm 1 scans the scalar from the right and reads "0"s at the end if $k_{\mathfrak{l}-1} = 0$. The "0"s read at the end does not change the value saved in $R[0]$ that is the correct computation result. In summary, Algorithm 1 can compute kP correctly with any input scalar $k \in [0, N-1]$.

Joye's RL m-ary algorithm satisfies the generality, implying that it can compute kP for any input $k \in \{0,1\}^{\mathfrak{l}}, k \in [0, N-1]$.

This proof will be given in the final version. We herein focus on the case of $m = 2$, which is shown in Algorithm 2.

Algorithm 1 Joye's double-add algorithm[6]

Input: $P \in E(\mathbb{F}_p), k = \sum_{i=0}^{l-1} k_i 2^i$
Output: kP
Uses: $R[0]$, $R[1]$
1: $R[0] \leftarrow O$
2: $R[1] \leftarrow P$
3: **for** $i = 0$ to $l - 1$ **do**
4: $\quad R[1 - k_i] \leftarrow 2R[1 - k_i] + R[k_i]$
5: **end for**
6: **return** R[0]

Algorithm 2 Joye's RL 2-ary algorithm [7]

Input: $P \in E(\mathbb{F}_p), k = \sum_{i=0}^{l-1} k_i 2^i$
Output: kP
Uses: A, $R[1]$, $R[2]$
Initialization
1: $R[1] \leftarrow O, R[2] \leftarrow O, A \leftarrow P$
Main Loop
2: **for** $i = 0$ to $l - 2$ **do**
3: $\quad R[1 + k_i] \leftarrow R[1 + k_i] + A, A \leftarrow 2A$
4: **end for**
Aggregation and Final correction
5: $A \leftarrow (k_{l-1} - 1)A + R[1] + 2R[2]$
6: $A \leftarrow A + P$
7: **return** A

3.2 Secure generality

We define the notion of the *secure generality* added to the generality as follows: If a scalar multiplication can compute kP regularly without dummy operations satisfying generality for $k \in [0, N - 1]$, where $N \in \{0, 1\}^l$ is the order of P, then we say that such an algorithm satisfies the *secure generality*.

Algorithm 2 executes the same computations of addition and doubling without any dummy operations for every bit of scalar yielding a point P and a scalar $k \in \{0, 1\}^l$. It is regular without dummy operations for any k, and thus satisfies secure generality. Algorithm 1 also executes the same computations of addition and doubling without any dummy operations until the final input bit of a scalar $k \in \{0, 1\}^l$. Its final step in the main loop becomes a dummy operation when processing $k_{l-1} = 0$. In fact, Algorithm 1 reads "0"s at the end if $k_{l-1} = 0$. Subsequently, the computation $R[1] \leftarrow 2R[1] + R[0]$ becomes a dummy operation, thus, we can know whether the scalar begins with "0" by changing the value of $R[1]$. If the result does not change, then the MSB of the scalar is "0". Thus, Algorithm 1 does not satisfy secure generality at the k_{l-1}.

3.3 Executable coordinate

Let us define the notion of a coordinate to a scalar multiplication algorithm. If the coordinate can be executed for an algorithm for $\forall k \in \{0, 1\}^l$, we say that a coordinate is *executable coordinate* for the algorithm. This notion is important because even if an algorithm satisfies secure generality, we must choose an executable coordinate.

Let us investigate the executable coordinates in Algorithm 1. Algorithm 1 requires addition or doubling formulae with O. This is why neither the affine nor Jacobian coordinate is executable.

Let us investigate Algorithm 2. Algorithm 2 contains exceptional inputs k. $R[1]$ and $R[2]$ are initialized as O in Step 2 and A is initialized as P in Step 4. In the main loop, $O + P$ appears independent of k in Step 6. It is obvious that $O + P$, $P + P$, and $-P + P$ are computed when $k = 1, 2, 0$ in the final correction, respectively. In summary, Algorithm 2 has to compute addition with O independent to k, $P+P$ if $k = 2$, $P-P$ if $k = 0$. Neither the affine nor Jacobian coordinate can compute all of $O + P$, $O + 2P$, $2P + 2P$, $P + P$, and $-P + P$. Meanwhile, the complete addition formulae [11] are executable coordinates. As shown in Section 2, we must sacrifice computational and memory complexity if we use the complete addition formulae.

We herein focus on Algorithm 2 as it satisfies the secure generality of k, and improve it such that it can be used for the affine coordinate that requires a small memory. It is noteworthy that our idea can be applied to Algorithms 1 easily and that Jacobian coordinate is also executable for our new Algorithms 7–8.

4. Variants of affine addition formulae

Affine addition formulae are advantageous because of less memory usage. The computational cost, however, depends on the ratio of inversion to the multiplication cost, where $t(A + A) = 2M + S + I$ and $t(2A) = 2M + 2S + I$.

Algorithm 3 Affine addition formula

Input: $P = (x_1, y_1)$ and $Q = (x_2, y_2)$
Output: P,$P + Q$
1: $t_0 \leftarrow (x_2 - x_1)^{-1}$
2: $y_2 \leftarrow y_2 - y_1$
3: $t_0 \leftarrow t_0 y_2$
4: $y_2 \leftarrow t_0^2 - x_1 - x_2$
5: $x_2 \leftarrow (x_1 - y_2)t_0 - y_1$
6: **return** (x_1, y_1),(y_2, x_2)

Algorithm 4 Affine doubling formula

Input: $P = (x_1, y_1)$
Output: P,$2P$
1: $t_0 \leftarrow 3x_1^2 + a$
2: $t_1 \leftarrow (2y_1)^{-1}$
3: $t_0 \leftarrow t_0 t_1$
4: $t_1 \leftarrow t_0^2 - 2x_1$
5: $t_2 \leftarrow (x_1 - t_1)t_0 - y_1$
6: **return** (x_1, y_1),(t_1, t_2)

The detailed algorithms are shown in Algorithms 3 and 4. It is noteworthy that both Algorithms 3 and 4 can retain the value of the input point of P, which can be used continually for the next input. Affine addition formulae have *exceptional points*. O can not be represented explicitly, while it is described as a point at infinity. Thus, affine addition formulae cannot compute $O + P = O$, $P - P = O$, or $2P = O$. The addition formula cannot compute $P + P$, which can only be computed by the doubling formula. When implementing affine addition formulae, branches are required to avoid such exceptional points. We want to fully utilize affine addition formulae because they reduce memory. Scalar multiplications should satisfy the generality

of k in Section 3, and thus suitable for any $k \in [0, N-1]$, where the order of P is N, which includes a special case of $k = 0$. Algorithm 2 satisfies the secure generality but the affine coordinate is not executable on them. Thus, we extend the affine addition formulae. The corresponding operations are shown in Algorithms 5 and 6, which can compute $P - P = O$ and $2P = O$ when $E(\mathbb{F}_p)$ does not include a point $(0, 0)$. For example, $E(\mathbb{F}_p)$ without two-torsion points, including the prime order elliptic curve on the Weierstrass form satisfy the condition. It is noteworthy that both Algorithms 5 and 6 retain the value of the input point of P similarly as Algorithms 3 and 4. Let us explain our idea of the extended affine addition formulae. The inversion of a $(mod\ p)$ can be computed by the extended Euclidean algorithm, $Ecd(a, p)$, or Fermat's little theorem, $Fermat(a, p) = a^{p-2}$ $(mod\ p)$. Interestingly, both algorithms can operate and output 0 even if $a = 0$; that is, both are executable for a special input of "0". Therefore, we compute $\frac{1}{x_2-x_1}$ and $\frac{1}{2y_1}$ from Algorithms 3 and 4 in the beginning and execute the remaining parts. Subsequently, the results for the ordinary inputs of P, Q are the same as those of Algorithms 3 and 4, respectively. Furthermore, the results for the exceptional inputs of $P-P$ and $2P = O$ can be given as $(0, 0)$, which is assumed as $O = (0, 0)$.

Algorithm 5 Extended affine addition

Input: $P = (x_1, y_1)$ and $Q = (x_2, y_2)$
Output: P,$P + Q$
1: $t_0 \leftarrow (x_2 - x_1)^{-1}$
2: $y_2 \leftarrow y_2 - y_1$
3: $x_2 \leftarrow x_2 - x_1$
4: $t_1 \leftarrow (x_2 + 2x_1)x_2$
5: $x_2 \leftarrow y_1 x_2$
6: $t_2 \leftarrow (y_2^2 t_0 - t_1)t_0$
7: $t_1 \leftarrow ((x_1 - t_2)y_2 - x_2)t_0$
8: **return** $(x_1, y_1), (t_2, t_1)$

Algorithm 6 Extended affine doubling

Input: $P = (x_1, y_1)$
Output: $P, 2P$
1: $t_0 \leftarrow 3x_1^2 + a, t_1 \leftarrow (2y_1)^{-1}$
2: $t_4 \leftarrow y_1^2, t_2 \leftarrow 8x_1 t_4$
3: $t_3 \leftarrow t_0^2 - t_2, t_2 \leftarrow t_1^2$
4: $t_3 \leftarrow t_3 t_2, x_1 \leftarrow x_1 - t_3$
5: $t_0 \leftarrow t_0 x_1, t_4 \leftarrow 2t_4$
6: $t_0 \leftarrow (t_0 - t_4)t_1$
7: $x_1 \leftarrow x_1 + t_3$
8: **return** (x_1, y_1),(t_3, t_0)

Neither Algorithm 3 nor 4 can output $P - P = (0, 0)$ or $2P = (0, 0)$, even if an inversion of $x_2 - x_1$ or $2y_1$ is computed by the Euclidean algorithm or Fermat's little theorem.

Theorem 1 Let $E(\mathbb{F}_p)$ be $y^2 = x^3 + ax + b$, $b \neq 0$ $(mod\ p)$, meaning that point $(0, 0)$ is not on $E(\mathbb{F}_p)$. P, Q are points on $E(\mathbb{F}_p)$. By setting $(0, 0)$ as O, the extended addition formula can compute the addition of P and Q correctly if $P \neq Q$ $(P \neq O, Q \neq O)$, $P - P = O$, and $O + O$. The extended doubling formula can compute the doubling of P correctly for any point on $E(\mathbb{F}_p)$.

Proof:

We can transform formulae (1) (2) to the extended affine addition formula by extracting the factor of $\frac{1}{X_2-X_1}$. When computing $P - P$, the inversion of zero must be computed. By the extended Euclidean algorithm, or Fermat's little theorem, we obtain zero for the inversion of zero. This demonstrates that by our affine addition formula, we can compute $P - P$:

$$X_3 = 0, Y_3 = 0 \tag{1}$$

This implies $P - P = (0, 0)$. Further, we regard $(0, 0)$ as O. Subsequently, our variant of affine addition formula computes $P - P = O$ correctly. Further, it is clear that $O + O = O$ can be computed correctly. We should emphasize that extracting the factor of $\frac{1}{x_2-x_1}$ does not affect the addition of other points because the factor $\frac{1}{x_2-x_1}$ will become zero only when computing $P - P$ and $O + O$, and in the other situation, extracting the factor of $\frac{1}{x_2-x_1}$ is always safe. The computational cost of Algorithm 5 is $6M + S + I$ and uses the memory of seven.

We can transform formulae (3) (4) to the extended affine doubling formula by extracting $\frac{1}{y_1}$. When computing $2P = O$, where P is of zero y-coordinate, the inversion of zero will be zero. Subsequently, we can compute $2P = (0, 0)$ by our affine doubling formula. Further, we regard $(0, 0)$ as O, implying that our variant of the affine doubling formula can compute $2P = O$ correctly when the point $(0, 0)$ is not on $E(\mathbb{F}_p)$. Further, extracting the factor of $\frac{1}{2y_1}$ does not affect the doubling of other points. The y-coordinate of P becomes zero only when $2P = O$. The variant of the affine doubling formula is exception-free, implying that it can compute the doubling of all points on $E(\mathbb{F}_p)$, where the point $(0, 0)$ is not on it. The computational cost of Algorithm 6 is $4M + 4S + I$ and uses the memory of seven.

It is noteworthy here that the original affine addition formulae cannot compute $P - P = O$, $P + O = P$, and $2P = O$, while our extended affine addition formulae can compute $P - P$ and $2P = O$ correctly. The Jacobian and projective addition formulae compute $P - P = O$ and $2P = O$ correctly. Thus, both coordinates become "executable coordinates" in our Algorithms 7–8. This implies that if our scheme perform well on the affine addition formulae to compute scalar multiplications, it can be extended to the Jacobian addition formulae or projective addition formulae easily and will perform better.

5. Secure and Efficient Elliptic Curve Scalar Multiplication

We propose memory-efficient algorithms that can avoid SCA by combining Algorithm 2 with the original and our extended affine addition formulae. It is noteworthy that the original affine coordinate is not executable for Algorithm 2 because the addition formula excludes $P+P$, $P+O$, and $P-P$ and the doubling formula excludes $2P$ with a two-torsion point P. We improve Algorithm 2 to avoid these exceptional inputs such that the original and extended affine coordinates become executable for Algorithm 2.

We also enhance the efficiency of our method by two-bit scanning using the affine double and quadruple formulae (DQ-formula) [9], which can compute both $2P$ and $4P$ simultaneously with only one inversion computation, denoted by $\{2P, 4P\} \leftarrow DQ(P)$. Thus, the computational cost

of obtaining both $2P$ and $4P$ in the affine coordinate is $t(\{2P, 4P\} \leftarrow P) = 8M + 8S + I$. Our primary idea to apply the DQ-formulae is by adjusting the length of the scalar by padding "0" in front of the scalar to guarantee no processing required for the remaining bits after a two-bit scanning. Using our adjusting idea, the processing of the remaining bits does not depend on the odd or even length of the input scalar k.

Algorithm 7 New 2-ary RL powering ladder

Input: $P \in E(\mathbb{F}_p)$
$k = \sum_{i=0}^{l-1} k_i 2^i, k \in [0, N]$
Output: kP
Uses: A, A[0], R[0], R[1]
Initialization
1: $R[0] = -P$
2: $R[1] = P$
3: $A \leftarrow 2P$
4: $R[k_0] \leftarrow R[k_0] + A$
Main Loop
5: **for** $i = 1$ to $l - 1$ **do**
6: $R[k_i] \leftarrow R[k_i] + A$
7: $A \leftarrow 2A$
8: **end for**
Final Correction
9: $R[k_0] \leftarrow R[k_0] - P$
10: $A \leftarrow -A + R[0] + 2R[1]$
11: **return** A

Algorithm 8 New two-bit 2-ary RL powering ladder

Input: $P \in E(\mathbb{F}_p)$
$k = \sum_{i=0}^{l-1} k_i 2^i, k \in [0, N]$
Output: kP
Uses: A, A[0], R[0], R[1]
Initialization
1: $R[0] = -P$
2: $R[1] = P$
3: $\{A, A[1]\} \leftarrow DQ(P) = \{2P, 4P\}$
4: $R[k_0] \leftarrow R[k_0] + A$
Main Loop
5: **for** $i = 1$ to $l - 1$ **do**
6: $R[k_i] \leftarrow R[k_i] + A$
7: $R[k_{i+1}] \leftarrow R[k_{i+1}] + A[1]$
8: $\{A, A[1]\} \leftarrow DQ(A[1])$
9: $i = i + 2$
10: **end for**
Final Correction
11: $R[k_0] \leftarrow R[k_0] - P$
12: $A \leftarrow -A + R[0] + 2R[1]$
13: **return** A

First, we improve Algorithm 2 to the new 2-ary RL Algorithm 7, and combine with two-bit scanning to obtain the new two-bit 2-ary RL Algorithm 8. Algorithms 7 and 8 consist of three parts: initialization, main loop and final correction. Compared to Algorithm 2, we change the initialization of $R[.]$ to avoid the exceptional initialization of O and the exceptional computation $O+P$ in the main loop. The initialization of $R[.]$ causes $R[1] + 2R[2] = O$ to be added to the final result in the aggregation of Algorithm 2. The initialization of $R[.]$ causes $R[0] + 2R[1] = P$ to be added to the final result in the final Step of our algorithms. Thus, we avoid the exceptional computations in the original final correction $A \leftarrow A + P$ of Algorithm 2. Steps 3 and 4 of Algorithms 7 and 8 help to avoid the exceptional computations of $P + P$ or $P - P$ if A is initialized as P. The final correction adjusts the excess computations in Steps 3 and 4 in Algorithms 7 and 8. We adjust the length of k to be even by padding "0" in front of input scalar k, and thus verify whether two-bit scanning can operate in Algorithm 8.

Next, we explain the affine coordinates (ordinary and our extended version) that is used in Algorithms 7 and 8. The original affine coordinate is used in Step1–9 of Algorithm 7 and Step 1–11 of Algorithm 8. Our extended affine formulae are used in Step 10 of Algorithm 7 and Step 12 of Algorithm 8. Our Algorithms 7 and 8 satisfy generality of k, and execute the same computations of addition and doubling without any dummy operations.

Theorem 2 proves that Algorithms 7–8 avoid all exceptional computations of affine addition formulae when $k \in [0, N - 3]$.

Theorem 2 Let $E/\mathbb{F}_p$ be an elliptic without two-torsion points. Let $E(\mathbb{F}_p) \ni P \neq O$ be an elliptic curve point, whose order is $N \in \{0, 1\}^l$. Then, Algorithms 7 and 8 can compute kP correctly for input $k \in [0, N - 3]$.

Proof:

We prove that all three parts exclude the exceptional computations of affine addition formulae, which are additions of $P \pm P$ and $O + P$, and doubling of $2P = O$. The doubling of $2P = O$ does not appear in the algorithms because of $E(\mathbb{F}_p)$ without two-torsion points. Thus, we only focus on the exceptional additions.

In the initialization, $R[0]$ and $R[1]$ initialized as $(P_x, -P_y)$ and (P_x, P_y) are "odd" scalar points such as $(2t + 1)P, t \in \mathbb{Z}$. A initialized as $((2P)_x, (2P)_y)$ is an "even" scalar point such as $2tP, t \in \mathbb{Z}$. It is obvious that $R[0] \leftarrow -P + 2P$ or $R[1] \leftarrow P + 2P$ in Step 4 is computed correctly by the addition formula if $N \neq 3$.

In the main loop, it is noteworthy that 1) $A \neq O$ because of $E(\mathbb{F}_p)$ without two-torsion points and A always increases as an "even" scalar point until $2^{l-1}P, 2^{l-1} < N$ when loop processing k_{l-2}. A increases to an "odd" scalar point at the end of loop. 2) Until loop processing k_{l-2}, $R[0] \neq O$ is always updated as an "odd" scalar point and with a smaller scalar than A. 3) Until loop processing k_{l-2}, $R[1] \neq O$ is also always updated as an "odd" scalar point. If $k = \{1\}^l$, $R[1]$ is always with a larger scalar than A and becomes $(2^{l-1} + 1)P, (2^{l-1} + 1) \leq N$. It also occurs when $k = N - 1$ or $k = N - 2$, so we excludes these two cases. Otherwise, $R[1]$ is with a smaller scalar than A in the main loop. In summary, $R[0], R[1], A[1] \neq O$ are scalar points of P whose scalars are never over N until loop processing k_{l-2}. Therefore, the "odd" scalar point can never be the same point as the "even" scalar point. The computations in the main loop exclude the exceptional computations of affine.

In the final correction, $R[k_0] \neq O$ is an "odd" scalar point and $-P = (N - 1)P$ is an "even" scalar point. Step 11 computes $P - P$ only when $k_0 = 0$. However we can always put an '0' in front of k to avoid this. If $k = 0$, Step12 computes the exceptional computation, $P - P$. Our extended affine addition formula can be used here because $E(\mathbb{F}_p)$ without two-torsion points excludes point $(0, 0)$.

The same proof can be shown in the two-bit scanning version.

6. Efficiency and Memory Analysis

We analyze the computational and memory complexity of Algorithms 2, 7 and 8, which are shown in Table 2. The memory complexity counts the number of $\mathbb{F}_p$ elements including the memory used in the addition formulae. The total computational complexity of Algorithm 2 with complete addition is $(\ell+1)24M$, if we ignore the computational complexity of ma, mb and A. Assuming the ratio of $S = 0.8M$, Algorithms 7 and 8 are more efficient than Algorithm 2 with complete addition if $\frac{I}{M} < 8.8$ and $\frac{I}{M} < 9.3$. Algorithm 8 is more efficient than Algorithm 7 if $\frac{I}{M} > 7.2$. In summary, if $9.3 > \frac{I}{M} > 7.2$, Algorithm 8 is the most efficient. If $\frac{I}{M} < 7.2$, Algorithm 7 is the most efficient.

As for memory complexity, Algorithms 7 and 8 can reduce that of Algorithm 2 with complete addition by 26% and 16%, respectively.

Table 2 Comparison Analysis

	Computational cost	memory
Algorithm 2 + Complete addition [11]	$(\ell+1)(24M + 6ma + 4mb + 46A)$	19
Algorithm 7 + Affine	$(6.4\ell + 18.8)M + (2\ell + 4)I$	14
Algorithm 8 + Affine	$(10\ell+33.2)M + \frac{3\ell+12}{2}I$	16

7. Conclusion

We proposed two new secure and compact elliptic curve scalar multiplication Algorithms 7 and 8 by combining Affine coordinates to Joye's regular RL 2-ary algorithm. Our primary ideas were to exclude the exceptional computations of $O+P$, $P-P=O$ and $P+P$ in the addition formulae from Joye's regular RL 2-ary algorithm and extend the Affine coordinates to compute $P-P=O$ and $2P = O$ by introducing a point $(0,0)$ as O when an elliptic curve $E(\mathbb{F}_p) \not\ni (0,0)$. Algorithm 8 combined two-bit scanning to further improve the efficiency. Consequently, Algorithms 7 and 8 were more efficient than Algorithm 2 with complete addition if $\frac{I}{M} < 8.8$ and $\frac{I}{M} < 9.3$. Further, Algorithms 7 and 8 could reduce the memory of Algorithm 2 with complete addition by 26% and 16%, respectively.

References

[1] Afreen, R. and Mehrotra, S.: A review on elliptic curve cryptography for embedded systems, *arXiv preprint arXiv:1107.3631* (2011).
[2] Bosma, W. and Lenstra, H. W.: Complete systems of two addition laws for elliptic curves, *Journal of Number theory*, Vol. 53, No. 2, pp. 229–240 (1995).
[3] Ebeid, N. and Lambert, R.: Securing the elliptic curve montgomery ladder against fault attacks, *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2009 Workshop on*, IEEE, pp. 46–50 (2009).
[4] Goundar, R. R., Joye, M., Miyaji, A., Rivain, M. and Venelli, A.: Scalar multiplication on Weierstraß elliptic curves from Co-Z arithmetic, *Journal of cryptographic engineering*, Vol. 1, No. 2, p. 161 (2011).
[5] Izu, T. and Takagi, T.: A fast parallel elliptic curve multiplication resistant against side channel attacks, *International Workshop on Public Key Cryptography*, Springer, pp. 280–296 (2002).
[6] Joye, M.: Highly regular right-to-left algorithms for scalar multiplication, *International Workshop on Cryptographic Hardware and Embedded Systems*, Springer, pp. 135–147 (2007).
[7] Joye, M.: Highly regular m-ary powering ladders, *International Workshop on Selected Areas in Cryptography*, Springer, pp. 350–363 (2009).
[8] Joye, M. and Yen, S.-M.: The Montgomery powering ladder, *International Workshop on Cryptographic Hardware and Embedded Systems*, Springer, pp. 291–302 (2002).
[9] Le, D.-P. and Nguyen, B. P.: Fast point quadrupling on elliptic curves, *Proceedings of the Third Symposium on Information and Communication Technology*, ACM, pp. 218–222 (2012).
[10] Miyaji, A. and Mo, Y.: How to enhance the security on the least significant bit, *International Conference on Cryptology and Network Security*, Springer, pp. 263–279 (2012).
[11] Renes, J., Costello, C. and Batina, L.: Complete addition formulas for prime order elliptic curves, *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, pp. 403–428 (2016).
[12] Susella, R. and Montrasio, S.: A Compact and Exception-Free Ladder for All Short Weierstrass Elliptic Curves, *International Conference on Smart Card Research and Advanced Applications*, Springer, pp. 156–173 (2016).
[13] Wroński, M.: Faster Point Scalar Multiplication on Short Weierstrass Elliptic Curves over Fp using Twisted Hessian Curves over Fp2, *Journal of Telecommunications and Information Technology* (2016).